CSE 522: Sublinear (and Streaming) Algorithms **Spring 2014**

Lecture 4: Finding Heavy Hitters in Streams

April 9, 2014

Lecturer: Paul Beame *Scribe: Paul Beame*

In the last lecture the stronger version of the algorithm reduced the space from $O(\varepsilon^{-2}(\log M + \log\log n))$ to roughly $O(\log M + \varepsilon^{-2}\log\log n)$. If the inputs are typical 64-bit word values, for example, then $\log_2 M$ is only 64 and is certainly larger than $\log n$. On the other hand, this might be comparable to or smaller than ε^{-2}; more generally we can think of the values of $\log M$ and $1/\varepsilon$ as typically polynomially related to each other so the smaller space algorithm as achieving a polynomial factor improvement in the space bound.

There is considerable practical utility in estimating the number of distinct values in the columns in a relation in order to estimate the costs of different query plans for relational databases; the differences in the costs for different plans can be very substantial, which makes these estimates very important. The algorithm of choice in practice is the HYPERLOGLOG algorithm of Flajolet et al. [**?**], which assumes the availability of a hash function h that behaves as if it were a truly random function. Because this hash function is assumed to be some standard procedure, its specification is not included in the space bound. The code for the algorithm is freely available. The basic idea also relies on the same zero counting method used in the other algorithms. It maintains these approximate counts in $K = \Theta(1/\varepsilon^2)$ separate buckets, and takes the harmonic mean of the results.

HYPERLOGLOG **Algorithm**

Initialize:
$K \leftarrow$ smallest power of 2 larger than $3/\varepsilon^2$
View random $h : [M] \to [M]$ as map $h : [M] \to [K] \times \{0,1\}^{m-\log_2 K}$
$z_1, \ldots, z_k \leftarrow 0$
Process:
for each i **do**
  $(k, j) \leftarrow h(x_i)$
  $z_k \leftarrow \max(z_k, \text{zeroes}(j))$
end for
Output:
$\alpha_K K^2 \cdot \text{HM}(2^{z_1}, \ldots 2^{z_k})$ where $\text{HM}(a_1, \ldots, a_K) = (\sum_{i=1}^{k} 1/a_i)^{-1}$ is the Harmonic Mean function and α_K is a specific constant depending on K.

The basic idea of the algorithm is that we expect F_0/K elements to land in each bucket so we expect each 2^{z_k} to be roughly F_0/K. We observe that $\text{HM}(F_0/K, \ldots, F_0/K) = (K(K/F_0))^{-1} = F_0/K^2$. When compared with the geometric or arithmetic means, the harmonic mean has the advantage of discounting outliers, particularly on the high side, where Markov's inequality gives weak bounds.

The α_K is given by a closed form integral that can be approximated to arbitrary precision and compensates for systematic bias from taking the harmonic mean so that the resulting estimator is unbiased. Flajolet et al. show that for a truly random function the resulting estimator has error at most $(1 + 1.48/\sqrt{K}) \leq 1 + \epsilon$ with high probability for a random function h.

A big advantage of the HYPERLOGLOG algorithm in practice is that each input element requires only one hash function evaluation. The space complexity ignoring the hash function is $O(\varepsilon^{-2} \log \log n)$.

1 Heavy Hitters in Data Streams

Consider the goal of finding all frequent elements in a data stream. In particular consider finding a *majority element*; i.e., an element j such that $f_j > 0$.

Majority Algorithm

Initialize: $c \leftarrow 0$
Process:
for each i **do**
  if c=0 **then**
    $j \leftarrow x_i$
  end if
  if $x_i = j$ **then**
    $c \leftarrow c + 1$
  else
    $c \leftarrow c - 1$
  end if
end for
Output: j

Claim: If $f_j > n/2$ then the output will be j:

Observe that while $c > 0$, the value of j does not change. We can therefore divide the input into segments based on when $c = 0$, and let $j_1, j_2, \ldots$ be the values of j during those segments. It is also clear that during the i-th segment, the value of j_i appears precisely $1/2$ the time among the inputs. Therefore if $f_j > n/2$, the last segment much end with $c > 0$ and the value must be j.

What if there is no j such that $f_j > n/2$? The value of j might be any input value. It might even occur only once in the input. One could verify this using a second pass over the input.

We will consider a more general view of finding frequent elements in data streams. The idea will be to compute an concise representation $\tilde{f}$ of an estimate for f satisfying:

- The space to represent $\tilde{f}$ is small.
- $\tilde{f}_j$ is "close" to f_j.
- $\tilde{f}_j$ is easy to compute given $\tilde{f}$.

Misra and Gries generalized that majority algorithm above to a more general method that we can think of in this form.

Misra-Gries Algorithm

Initialize:
$A \leftarrow \emptyset$, A is a set of up to $k-1$ pairs $(j, \tilde{f}_j)$.
Process:
for each i **do**
  if $x_i \in A$ **then**
    $\tilde{f}_{x_i} \leftarrow \tilde{f}_{x_i} + 1$
  else if $|A| < k-1$ **then**
    Add x_i to A
    $\tilde{f}_{x_i} \leftarrow 1$
  else
    for each $j \in A$ **do**
      $\tilde{f}_{x_i} \leftarrow \tilde{f}_{x_i} - 1$
      if $\tilde{f}_j = 0$ **then**
        Remove j from A
      end if
    end for
  end if
end for
Output: $\tilde{f} \leftarrow A$
$\tilde{f}_j$ is as given for $j \in A$, $\tilde{f}_j = 0$ if $j \notin A$.

Claim: $f_j - \frac{n}{k} \leq \tilde{f}_j \leq f_j$ for all j.

Proof. We think of the algorithm as maintaining a current estimate $\tilde{f}_j$ for all $j \in [M]$, where $\tilde{f}_j = 0$ for $j \notin A$.

Clearly $\tilde{f}_j \leq f_j$ since each occurrence of j can increase $\tilde{f}_j$ by at most 1.

Observe that we can rewrite the algorithm in equivalent form by saying that, whenever a $j' \not\in A$ is encountered when $|A| = k-1$, we first set $\tilde{f}_{j'}$ to 1, and then subtract 1 from all k positive values of $\tilde{f}_j$ including $\tilde{f}_{j'}$.

With this view, the value of $\tilde{f}_j$ is equal to f_j minus the number of times that line 12 is executed with $\tilde{f}_j > 0$. We can allocate each such execution with specific occurrences of the k distinct elements in the original data stream (including j) that have positive $\tilde{f}$ values. Therefore, the number of times that this can occur is at most a $1/k$ fraction of the total length of the stream. □

Corollary 1.1. $A \supseteq \{j \ : \ f_k > n/k\}$

This implies that the algorithm has detected all heavy hitters, though being in the set A is no guarantee.

Measuring the quality of $\tilde{f}$ estimates: For $f = (f_1, \ldots, f_M)$ we write the ℓ_p norm of f as:

$$||f||_p = \left(\sum_{j=1}^{M} |f_j|^p \right)^{1/p}.$$

Because it will sometimes be useful for expressing properties of the algorithms we consider, for convenience we also define $f_{-j} = (f_1, \ldots, f_{j-1}, f_{j+1}, \ldots, f_M)$. Observe that $||f||_1 = n$ and $||f||_2$ is the usual Euclidean norm. Note that the $||\cdot||_p$ decreases with increasing p. All the norms are equal if there is only one element j with $f_j > 0$. However, if the input consists of n distinct elements then $||f||_2 = \sqrt{n}$ and $||f||_p$ approaches 1 as $p \to \infty$. Therefore approximations in terms of $||\cdot||_p$ for larger p are preferred,

If we choose $k = \lceil 1/\varepsilon \rceil$ in the Misra-Gries algorithm then we see that we obtain

$$\text{Misra-Gries estimate: } f_{j^*} - \varepsilon||f||_1 \leq \tilde{f}_{j^*} \leq f_{j^*} \text{ for every } j^* \in [M].$$

This is useful for determining heavy hitters.

Definition 1.2. *j is a (γ, p)-heavy hitter iff*

$$f_j \geq \gamma \cdot ||f||_p.$$

The Misra-Gries algorithm then yields an approximate algorithm to find all $(\gamma, 1)$-heavy hitters for any $\gamma > \varepsilon$: It will simply output all $j \in A$ such that $\tilde{f}_j \geq (\gamma - \varepsilon)n$.

The resulting algorithm will report every j with $f_j \geq \gamma \cdot ||f||_1$, it will not report any j such that $f_j < (\gamma - \varepsilon)||f||_1$ and may or may not report any j with $(\gamma - \varepsilon)||f||_1 \leq f_j < \gamma \cdot ||f||_1$. Observe that the space of this algorithm is $O(\frac{1}{\varepsilon}(\log M + \log n))$. This algorithm is particularly nice because it does not require any randomness.

2 Sketching for Heavy Hitters

Definition 2.1. *A function* sk *on data streams in* $[M]^*$ *is called a* sketch *iff there is a space-efficient combining algorithm* COMB *such that for all* $\sigma_1, \sigma_2 \in [M]^*$, $\mathrm{COMB}(\mathrm{sk}(\sigma_1), \mathrm{sk}(\sigma_2)) = \mathrm{sk}(\sigma_1\sigma_2)$.

We will discuss a different algorithm called the COUNT-MIN Sketch due to Cormode and Muthukrishnan which is a randomized algorithm for heavy hitters. We first analyze a simple sketch on which it will be based.

Simple Hash Count Sketch

Initialize: $k \leftarrow \lceil 2/\varepsilon \rceil$
$C \leftarrow$ length k integer array, initially 0
Choose $h : [M] \to [k]$ from a 2-universal family of hash functions
Process:
for each i **do**
    $C[h(x_i)] \leftarrow C[h(x_i)] + 1$
end for
Output: $\tilde{f} \leftarrow (C, h)$
$\tilde{f}_j = C[h(j)]$

We can assume without loss of generality that $\varepsilon \geq 1/n$ and hence $\log k$ is $O(\log n)$. The total space of the sketch is $O(\frac{1}{\varepsilon}\log n + \log M))$ since it takes $O(\log M + \log k) = O(\log M + \log n)$ bits to specify h and at most $k \log n$ bits to record C.

Fix $j^* \in [M]$. $C[h(j^*)]$ is incremented once for each i such that $x_i = j^*$ and never is decremented; therefore $\tilde{f}_{j^*} = C[h(j^*)] \geq f_{j^*}$. Write

$$Y_j = \begin{cases} 1 & \text{if } h(j) = h(j^*) \\ 0 & \text{otherwise.} \end{cases}$$

Then $\tilde{f}_{j^*} = \sum_{i=1}^{n} Y_{x_i} = \sum_{j=1}^{M} f_j Y_j$. Observe that $Y_{j^*} = 1$ and for $j \neq j^*$, $\mathbb{E}(Y_j) = 1/k$ since h is 2-universal. Let $X = \tilde{f}_{j^*} - f_{j^*} = C[h(j^*)] - f_{j^*}$. Then

$$\begin{aligned} \mathbb{E}(X) = \mathbb{E}(\tilde{f}_{j^*} - f_{j^*}) &= \sum_{j \neq j^*} f_j \mathbb{E}(Y_j) \\ &= \sum_{j \neq j^*} f_j/k \\ &= \frac{||f||_1 - f_{j^*}}{k} = \frac{||f_{-j^*}||_1}{k} \end{aligned}$$

By Markov's inequality we have

$$\mathbb{P}[X \geq \varepsilon ||f_{-j^*}||_1] \leq \frac{\mathbb{E}(X)}{\varepsilon ||f_{-j^*}||_1} = \frac{1}{\varepsilon k} \leq \frac{1}{2}.$$

Therefore, any single $\tilde{f}_{j^*}$ is correct within an additive $\varepsilon||f_{-1}||_1$ of f_{j^*} probability at least 1/2.

The basic idea of the COUNT-MIN Sketch will be to reduce the error by running $t = \lceil \log_2(1/\delta)$ independent copies of the Simple Hash Sketch in parallel and taking the minimum of the answers. Before we describe the algorithm, we will consider some extensions of the basic streaming model that this extension can handle.

Cash Register and Turnstile models of Streaming We can imagine a short-hand expression for multiple copies of a single element by writing (x_i, c_i) for integer c_i to express a sequence of c_i copies of x_i in the input. One can imagine this as an insertion of these c_i copies. We can extend this notion to allow deletion by allowing negative values for c_i. With these notions we write

$$f_j = \sum_{i:\ x_i = j} c_i.$$

If the c_i are all positive, then this extension is known as the *cash register* model; if both positive and negative values are possible, it is known as the *turnstile* model.

The COUNT-MIN Sketch will work for the cash register model and indeed any input in the turnstile model in which all f_j are non-negative.

The COUNT-MIN Sketch Algorithm

Initialize:
$k \leftarrow \lceil 2/\varepsilon \rceil$
$t \leftarrow \lceil \log_2(1/\delta) \rceil$
$C \leftarrow t \times k$ integer array, initially 0
Choose $h_1, \ldots h_t : [M] \rightarrow [k]$ independently from a 2-universal family of hash functions
Process:
for each i **do**
    for $s = 1$ to t **do**
        $C[s, h_s(x_i)] \leftarrow C[s, h_s(x_i)] + c_i$
    end for
end for
Output: $\tilde{f} \leftarrow (C, h_1, \ldots, h_t)$
$\tilde{f}_j = \min\{C[s, h_s(j)] \ :\ s = 1, \ldots, t\}$

The total space of the sketch is $O(\frac{\log(1/\delta)}{\varepsilon}(\log n + \log M))$.

Fix $j^* \in [M]$ and assume that $f_j \geq 0$ for all j. Write $X_s = C[s, h_s(j^*)] - f_{j^*}$ for $s = 1, \ldots, t$. Since each $f_j \geq 0$, the contribution of every element j other than j^* to $C[s, h_s(j^*)]$ is non-negative, we have $X_s \geq 0$ for all s and hence $\tilde{f}_{j^*} \geq f_{j^*}$. Observe that the expectation of each X_s is precisely

the same as that of X for the simple hash algorithm, namely $\mathbb{E}(X_s) = ||f_{-j^*}||_1/k$. Therefore

$$\begin{aligned}
\mathbb{P}[\tilde{f}_{j^*} - f_{j^*} \geq \varepsilon||f_{-j^*}||_1] &= \mathbb{P}[\min(X_1, \ldots, X_t) \geq \varepsilon||f_{-j^*}||_1] \\
&= \mathbb{P}[\bigwedge_{s=1}^{t}(X_s \geq \varepsilon||f_{-j^*}||_1)] \\
&= \prod_{s=1}^{t}\mathbb{P}[X_s \geq \varepsilon||f_{-j^*}||_1] \\
&\leq \frac{1}{2^t} \leq \delta.
\end{aligned}$$

Therefore $f_{j^*} \leq \tilde{f}_{j^*} \leq f_j + \varepsilon||f_{-j^*}||_1$ except with probability δ.

Note that the approximation error in this estimate is comparable to that of the Misra-Gries algorithm (except for the minor difference in removing the occurrences of the element from the error bound and the fact that it is an over-estimate rather than an under-estimate). Note also that the above error is on a per-value basis rather than overall. In order to get an overall probability of error at most δ one would need to use $t = \log_2(1/\delta) + \log_2 M$.

The main reason one might prefer the COUNT-MIN Sketch algorithm is that it is a linear sketching algorithm that allows deletions as well as insertions (though it does require that the final frequencies are all non-negative).